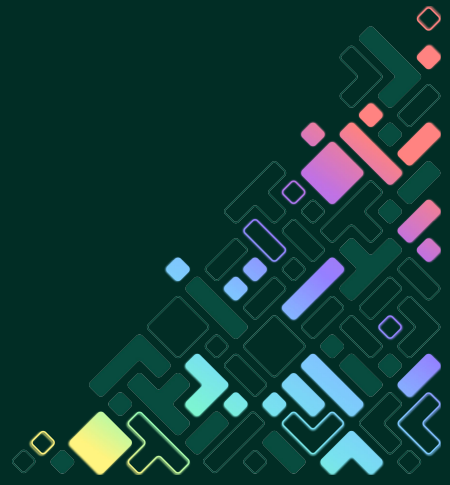
EMBEDDED OPEN SOURCE SUMMIT

Standardizing the generation and signing of boot images

2024-04-17, Seattle

Neha Malcom Francis, TI
Simon Glass, Google
Vignesh Raghavendra, TI



THE LINUX FOUNDATION

#EmbeddedOSSummit

Introduction to the Speakers

```
Neha-Malcom-Francis {
      software-engineer;
      Texas-Instruments;
      location = “Bangalore, KA, India”;
};
```

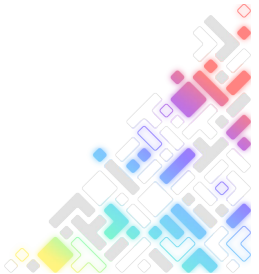
```
Simon-Glass {
      software-engineer;
      Google;
      location = “Boulder, CO, USA”;
};
```

```
Vignesh-Raghavendra {
      software-engineer;
      Texas-Instruments;
      location = “Bangalore, KA, India”;
};
```

Motivation

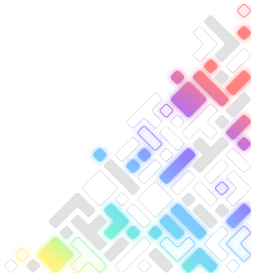
- Bootloaders of modern SoCs do more than just SDRAM/DDR initialization
- Can have multiple platform level firmwares to load and start
- Requires complex packing and signing procedures
- Over the years bootloaders like U-Boot has solved this in multiple ways

This talk is about exploring standardization efforts in U-Boot and how it can leveraged across Linux ecosystem



Overview

- Quick intro to U-Boot
- Firmware packing: Modern Systems
- Boot flow of complex SoCs
- Binman!
- Extending Binman to support TI K3 SoCs
- Future development



Das U-Boot: The Universal Boot-loader

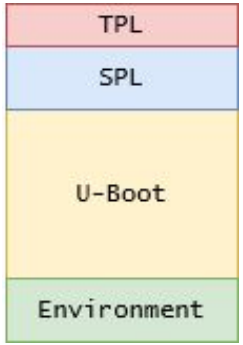
- Open source bootloader for embedded devices
- Rich set of peripheral drivers and stacks
- Tightly integrated with the Linux kernel
- Multiple Architectures: ARM, x86, RISC-V etc.
- Supports Device Tree based HW description

Firmware packing - Ancient

- Old approach: make
- Set location of environment: flash!

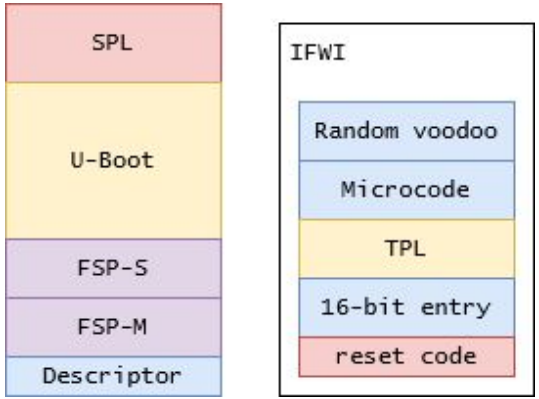
Firmware packing - Legacy

- Multi stage boot
  - SPL and TPL come along
- cat them?
- What about additional firmwares
  - TF-A? FPGA FW?



Firmware packing - Modern systems

- x86 Chromebooks:
  - Various binary blobs
  - Needs 32-bit code to run FSP
  - IFWI



A typical ARM64 Boot Flow

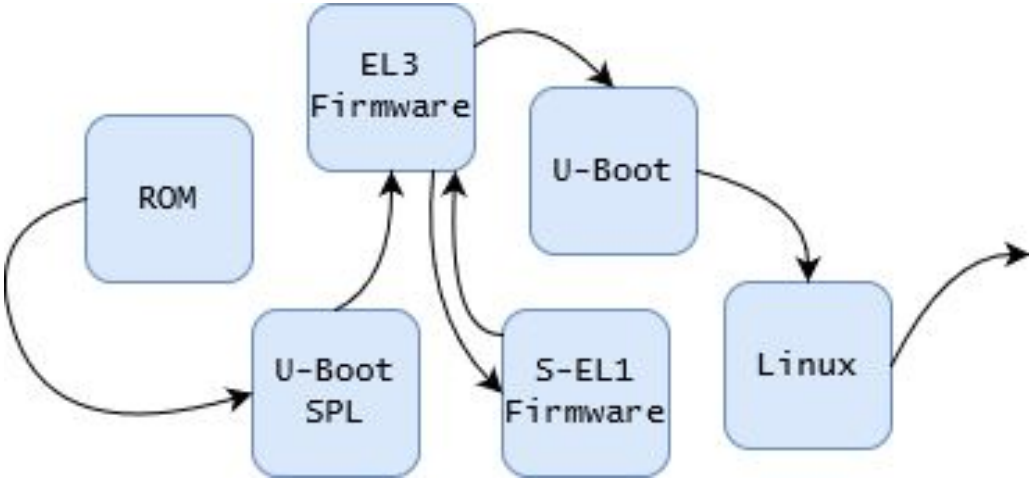
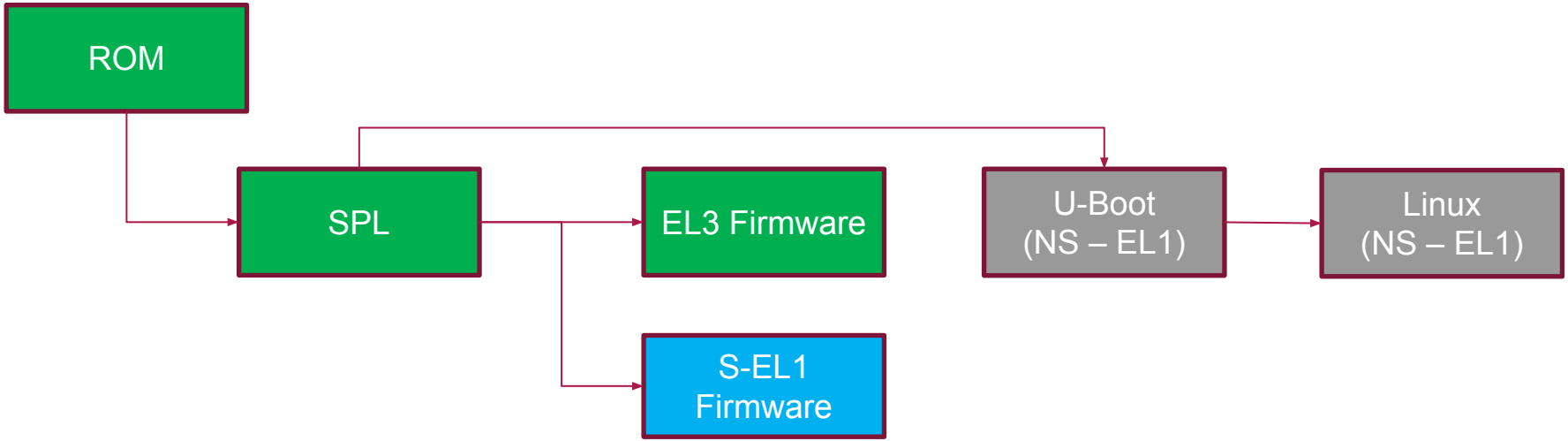


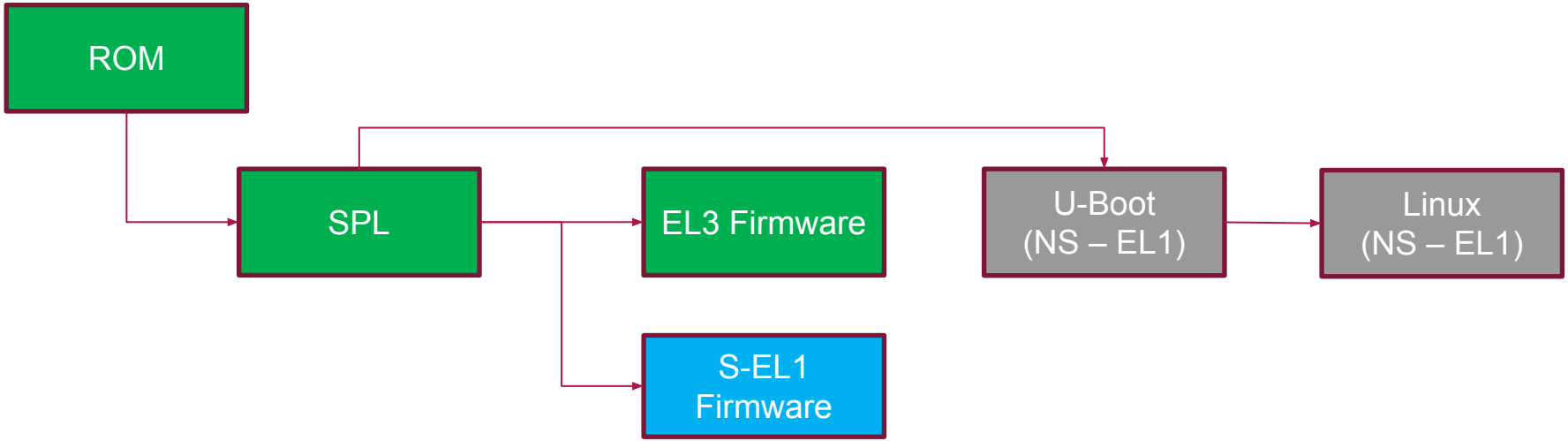
Image courtesy Open Source Summit Europe 2022 Bootloaders-101 - Bryan Brattlof

ARM64: Typical Image load Sequence



NS: Non Secure
S: Secure

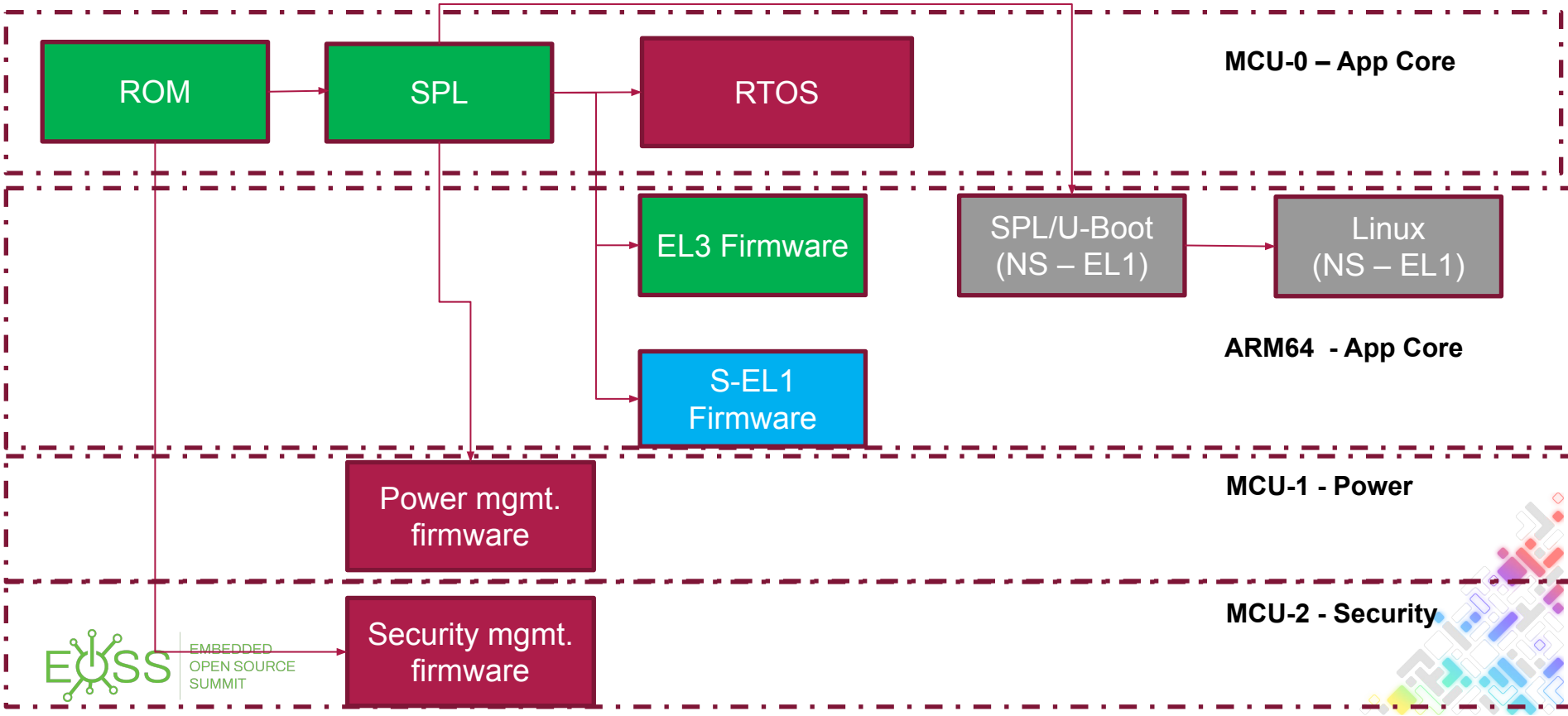
ARM64: Typical Image load Sequence



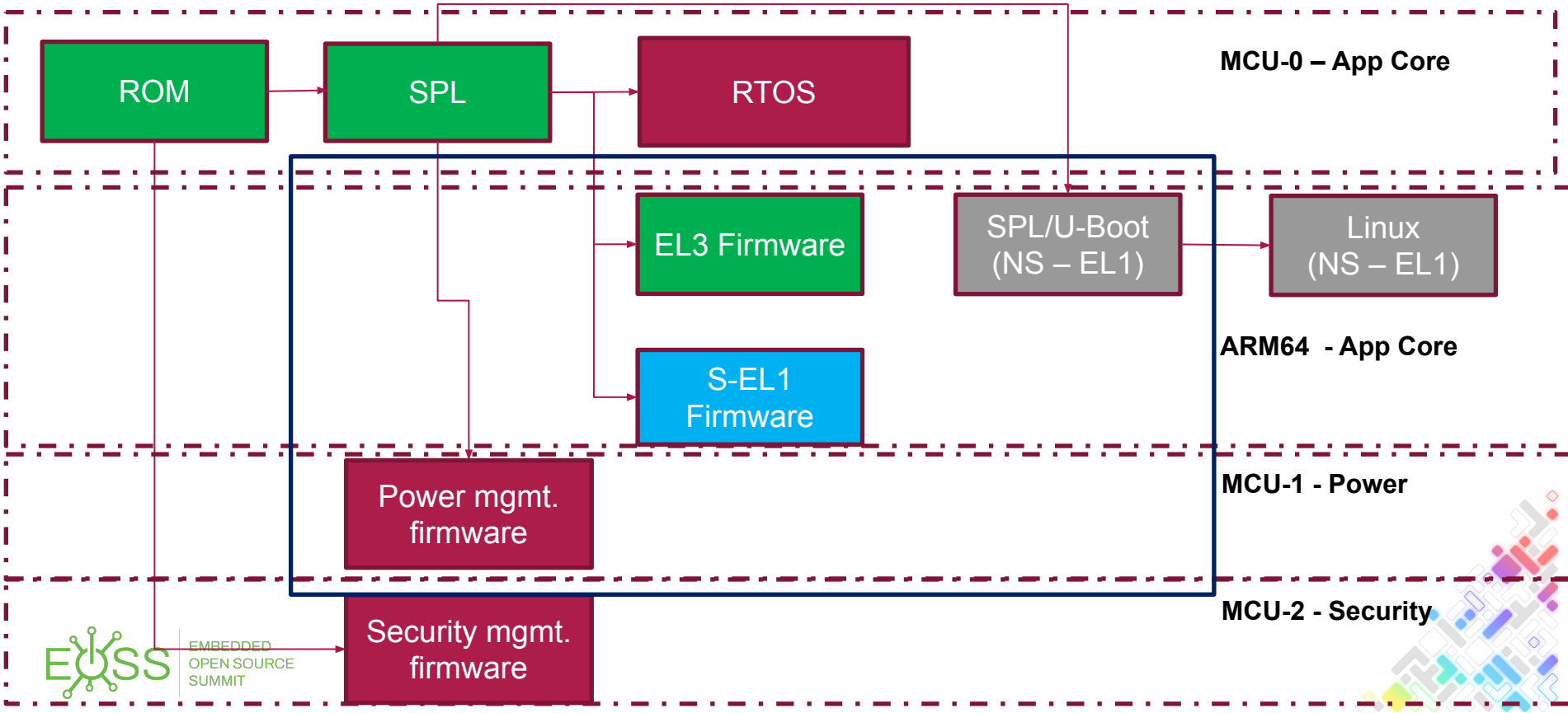
How about heterogenous SoCs?

NS: Non Secure
S: Secure

Loading AMP cores



Loading AMP cores



TI K3 Boot Flow - Heterogenous SoCs

- Multiple MCU cores and MPUs on same die with ROM typically on MCU
- Need to boot R5 (32-bit) and A72 (64-bit) cores
- First SPL runs on primary boot core (R5)
- Another SPL stage on A72

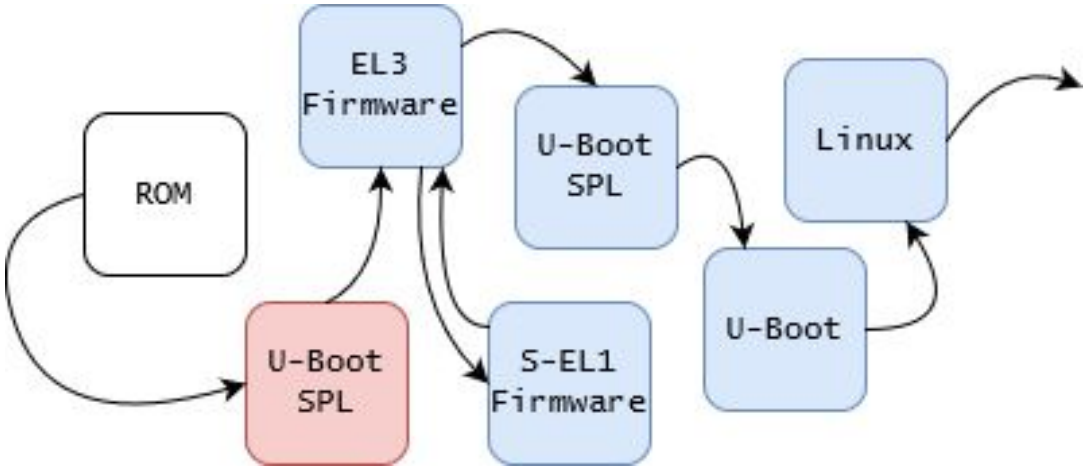
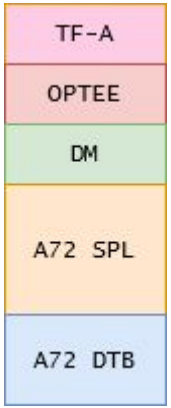


Image courtesy Open Source Summit Europe 2022 Bootloaders-101 - Bryan Brattlof

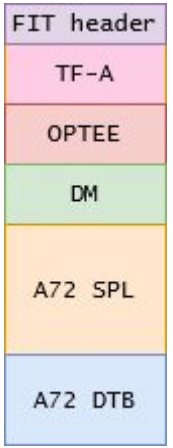
TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A53/A72/...) Bootloader
  - SPL stage image built from U-Boot source
- Firmwares from external source
  - Device Management (DM) Firmware running on dedicated MCU (TI version of ARM System Control Processor)
  - ARM secure world firmware
    - Trusted Firmware-A (TF-A) at EL3
    - Optional OPTEE at S-EL1



TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A72) Bootloader
  - SPL, DM firmware, TF-A, OPTEE
- FIT (Flattened uImage Tree)
  - Popular image packing method
  - mkimage + Device Tree like description language



TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A72) Bootloader
  - SPL, DM firmware, TF-A, OPTEE
- FIT
- Add Security/ Authenticated boot
  - SoC and End user specific X.509 Certificate or such other certificate
  - Optional Encryption
  - Signing using public key cryptography methods (RSA, ECDSA etc)

TI K3 Boot-loader binaries: a firmware tangle!

- Application core (A72) Bootloader
  - SPL, DM firmware, TF-A, OPTEE
- FIT
- Add Security/ Authenticated boot
  - SoC and End user specific X.509 Certificate
  - Encryption
  - Signing using public key cryptography methods (RSA, ECDSA etc)
- All the above makes up single container **(tispl.bin)** to be loaded by ROM / MCU bootloader starting Application core

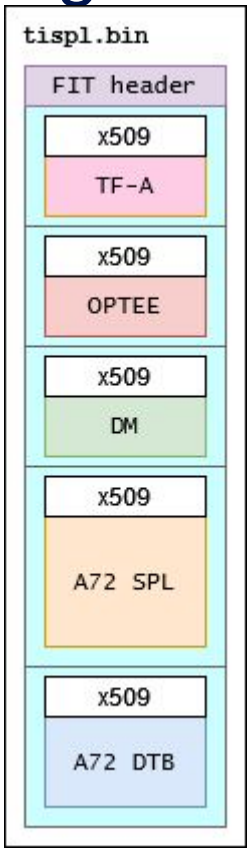
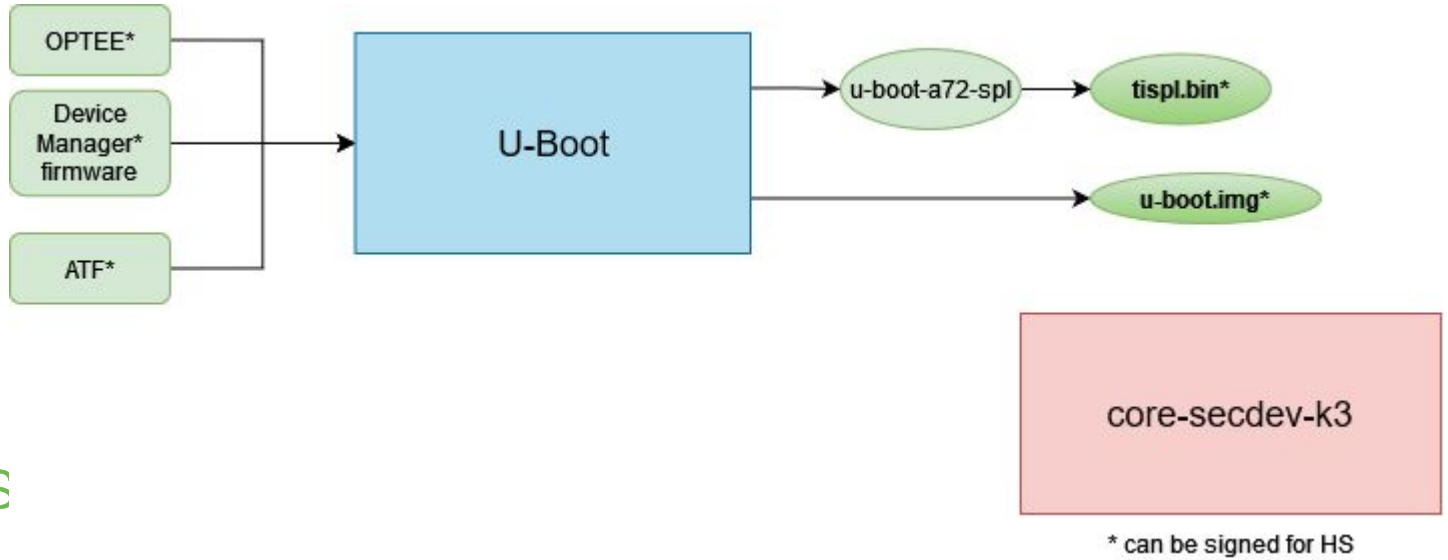


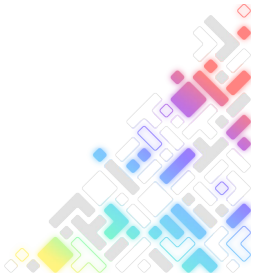
Image generation: Custom scripts

- Complex steps
  - **Pack**: Makefile and mkImage tools
  - **Sign**: Vendor specific custom scripts. Eg.: core-secdev-k3: certificate creation and signing tools



Custom scripts: Issues

- Maintaining and scaling
  - Boot flow variation (Presence or absence of MCU core firmwares)
  - Multiple device type (Non Secure Device vs High Secure Device)
  - Ability for end users to understand and play with the boot images
- Moving away from Vendor signing scripts process is difficult
- Not friendly for all distro builds
- No unit level testing



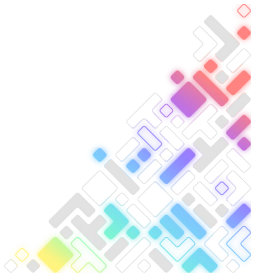
TI K3 Boot-loader binaries: a firmware tangle!

```
# build image

cmd_k3_mkits = \
        $(srctree)/tools/k3_fit_atf.sh \
        $(CONFIG_K3_ATF_LOAD_ADDR) \
        $(patsubst %,$(obj)/dts/%.dtb,$(subst ",,$(LIST_OF_DTB))) > $@

 # Get input file info
HS_SHA_VALUE=$(openssl dgst -sha512 -hex $INPUT_FILE | sed -e "s/^.*= //g")
HS_IMAGE_SIZE=$(cat $INPUT_FILE | wc -c)

# Get software revision info
HS_SWRV=$(cat ${PREFIX}/keys/swrv.txt)

# Parameters to get populated into the x509 template
HS_SED_OPTS="-e s/TEST_IMAGE_LENGTH/${HS_IMAGE_SIZE}/ "
HS_SED_OPTS+="-e s/TEST_IMAGE_SHA_VAL/${HS_SHA_VALUE}/ "
HS_SED_OPTS+="-e s/TEST_SWRV/${HS_SWRV}/ "
TMPX509=$(mktemp) || exit 1
cat ${PREFIX}/templates/x509-template.txt | sed ${HS_SED_OPTS} > ${TMPX509}

# Generate x509 certificate
TMPCERT=$(mktemp) || exit 1

openssl req -new -x509 -key ${PREFIX}/keys/custMpk.pem -nodes -outform DER -out
${TMPCERT} -config ${TMPX509} -sha512

# Append x509 certificate
cat ${TMPCERT} $INPUT_FILE > $OUTPUT_FILE


#Sign image
cmd_k3secureimg = $(TI_SECURE_DEV_PKG)/scripts/secure-binary-image.sh \
        $< $@ \
        $(if $(KBUILD_VERBOSE:1=), >/dev/null)
```

Why is packaging so hard?

- Collection of inputs
- Dependencies between entries
- Alignment
- Signing
- Run-time discovery of content
- Code/data-size limitations
- Compression

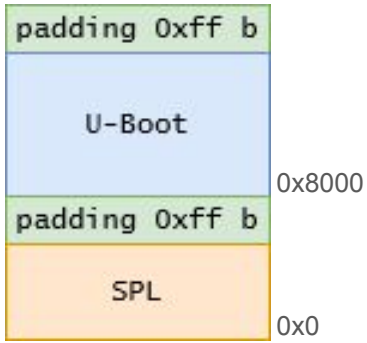
- Formats (FIT, CBFS, FIP)
- Processing time
- SoC-specific tools
- Examining an image / map
- Changing an image later
- Size constraints on parts
- Split over several phases



Binman

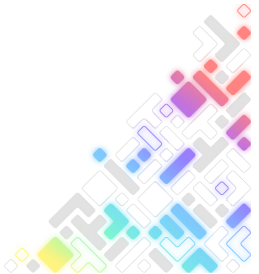
- Reimagining the image as **described data**
- Use DT language for image description as frontend
- Python backend
- Currently hosted under U-Boot project

```
binman {
    size = <0x100000>;
    pad-byte = <0xff>;
    u-boot-spl {
    };
    u-boot {
        offset = <0x8000>;
    };
};
```



When does Binman run?

- Initially, as part of the U-Boot build
  - After all inputs have been built
  - It packages the inputs
- Binman can be also be run later
  - With the same inputs and description file
  - Allows signing with different keys
- There CLI utility to run at any time

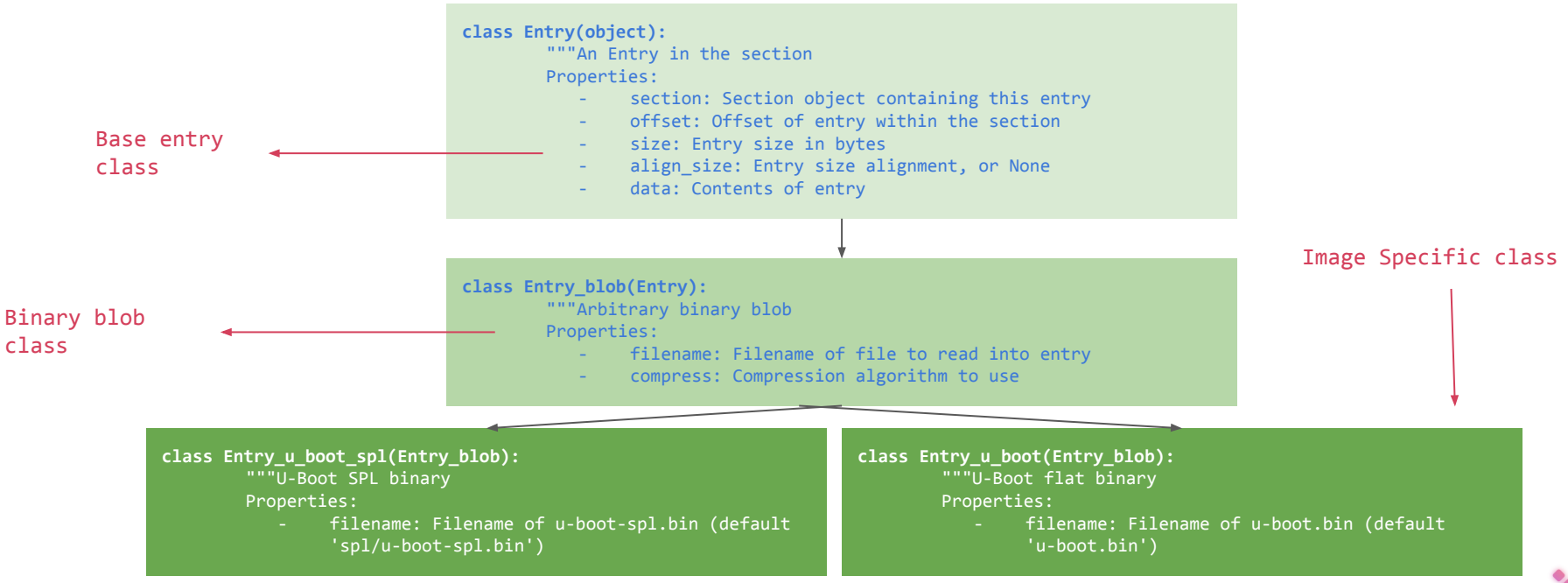


Binman - entries

- Images consist of 'entries'
  - Each entry holds a binary or some text or other data
  - Each entry has properties
- Entries are packed one after the other, in order
  - Unless entries have explicit offsets
- Entries cannot overlap
- Hierarchical
  - Entries can have child entries

```
// U-Boot SPL entry
u-boot-spl {
};


// U-Boot entry
u-boot {
    offset = <0x8000>;
};
```

Binman - entries

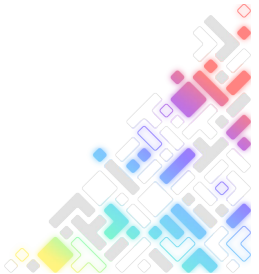



```
// U-Boot SPL entry
u-boot-spl {
};
```

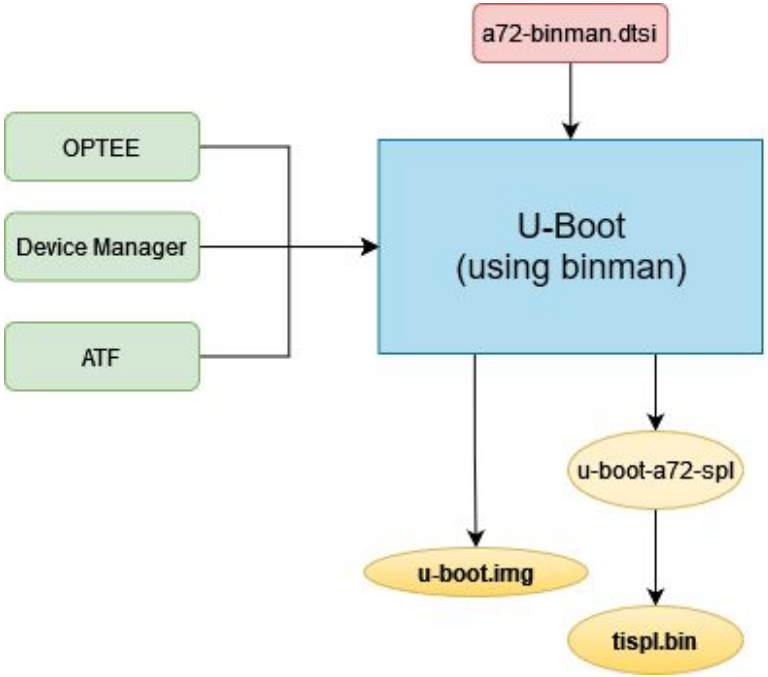
```
// U-Boot entry
u-boot {
        offset = <0x8000>;
};
```

Binman - adding an entry type

- Add a new <entry-name>.py to the etype/ directory
  - Define Entry_ti_secure class
  - Binman will find it
- Entries can run (optionally build) external tools and arbitrary scripts.
  - `tools.Run(...)`
- Main logic is in control.py (ProcessImage())
- Code has lots of comments
- Look at other entries for ideas
- Make sure to add a test!

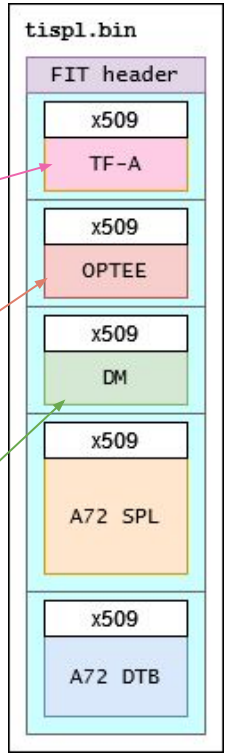


Extending Binman to support K3



Binman representation - tispl.bin

```
&binman {
        tispl {
                filename = “tispl.bin”;
                pad-byte = <0xff>;
                fit {
                        description = “FIT configuration to load ATF and SPL”;
                        #address-cells = <1>;
                        images {
                                atf {
                                        ti-secure {
                                                content = <&atf>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        atf: atf-bl31 {
                                        };
                                };

                                tee {
                                        ti-secure {
                                                content = <&tee>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        tee: tee-os {
                                        };
                                };

                                dm {
                                        ti-secure {
                                                content = <&dm>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        dm: blob-ext {
                                                filename = “ti-dm.bin”;
                                        };
                                };
```

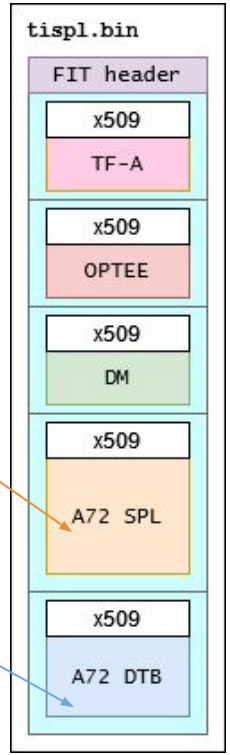


Binman representation - tispl.bin

```
fdt-0 {
        ti-secure {
                content = <&u_boot_spl_nodtb>;
                keyfile = "custMpk.pem";
        };
        u_boot_spl_nodtb: u-boot-spl-nodtb {
        };
};
```

```
fdt {
        ti-secure {
                content = <&spl_dtb>;
                keyfile = "custMpk.pem";
        };
        spl_dtb: u-boot-spl-dtb {
        };
};
```

```
                };
            };
        };
};
```



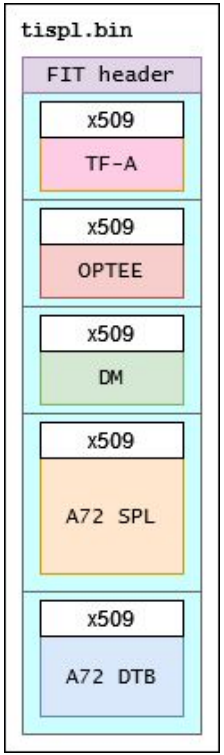
Binman representation - tispl.bin

```
&binman {
        tispl {
                filename = “tispl.bin”;
                pad-byte = <0xff>;
                fit {
                        description = “FIT configuration to load ATF and SPL”;
                        #address-cells = <1>;
                        images {
                                atf {
                                        ti-secure {
                                                content = <&atf>;
                                                keyfile = “custMpk.pem”;
                                                load = <CONFIG_K3_ATF_LOAD_ADDR>;
                                        };
                                        atf: atf-bl31 {
                                        };
                                };
                                tee {
                                        ti-secure {
                                                content = <&tee>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        tee: tee-os {
                                        };
                                };
                                dm {
                                        ti-secure {
                                                content = <&dm>;
                                                keyfile = “custMpk.pem”;
                                        };
                                        dm: blob-ext {
                                                filename = “ti-dm.bin”;
                                        };
                                };
```

Custom etype

CONFIG option

Standard etype



TI K3 Boot-loader binaries: a firmware tangle!

```
# build image

cmd_k3_mkits = \
        $(srctree)/tools/k3_fit_atf.sh \
        $(CONFIG_K3_ATF_LOAD_ADDR) \
        $(patsubst %,$(obj)/dts/%.dtb,$(subst ",,$(LIST_OF_DTB))) > $@

 # Get input file info
HS_SHA_VALUE=$(openssl dgst -sha512 -hex $INPUT_FILE | sed -e "s/^.*= //g")
HS_IMAGE_SIZE=$(cat $INPUT_FILE | wc -c)

# Get software revision info
HS_SWRV=$(cat ${PREFIX}/keys/swrv.txt)

# Parameters to get populated into the x509 template
HS_SED_OPTS="-e s/TEST_IMAGE_LENGTH/${HS_IMAGE_SIZE}/ "
HS_SED_OPTS+="-e s/TEST_IMAGE_SHA_VAL/${HS_SHA_VALUE}/ "
HS_SED_OPTS+="-e s/TEST_SWRV/${HS_SWRV}/ "
TMPX509=$(mktemp) || exit 1
cat ${PREFIX}/templates/x509-template.txt | sed ${HS_SED_OPTS} > ${TMPX509}

# Generate x509 certificate
TMPCERT=$(mktemp) || exit 1

openssl req -new -x509 -key ${PREFIX}/keys/custMpk.pem -nodes -outform DER -out
${TMPCERT} -config ${TMPX509} -sha512

# Append x509 certificate
cat ${TMPCERT} $INPUT_FILE > $OUTPUT_FILE


#Sign image
cmd_k3secureimg = $(TI_SECURE_DEV_PKG)/scripts/secure-binary-image.sh \
        $< $@ \
        $(if $(KBUILD_VERBOSE:1=), >/dev/null)
```

ti-secure - Example of etype

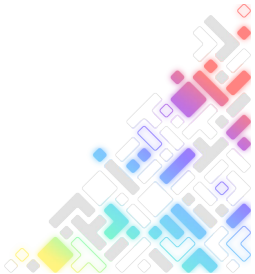
```
class Entry_ti_secure(Entry_x509_cert):
      def __init__(self, section, etype, node):
            super().__init__(section, etype, node)
            self.openssl = None
            ...
      def ReadNode(self):
            super().ReadNode()
            self.key_fname = self.GetEntryArgsOrProps([
                        EntryArg('keyfile', str)], required=True)[0]
            self.sha = fdt_util.GetInt(self._node, 'sha', 512)
            ...
```

ti-secure

```
class Entry_ti_secure(Entry_x509_cert):
      ...
      def ObtainContents(self):
            data = self.data
            if data is None:
                  data = self.GetCertificate(False)
            if data is None:
                  return False
            self.SetContents(data)
            return True

    def ProcessContents(self):
            # The blob may have changed due to WriteSymbols()
            data = self.data
            return self.ProcessContentsUpdate(data)
```

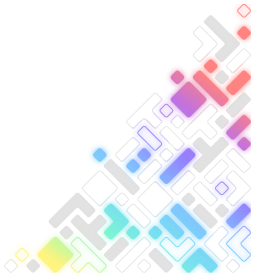
ti-secure

```
class Entry_ti_secure_rom(Entry_x509_cert):
        …
        def AddBintools(self, btools):
                super().AddBintools(btools)
                self.openssl = self.AddBintool(btools, 'openssl')
```

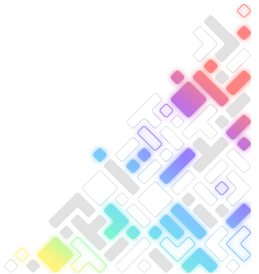
Advantages

- Data driven representation (FIT format)
  - Intuitive to read
- Object oriented design using Python
  - Abstracts generic packing script
  - Vendor specific scripts encapsulated within Python class
- Templates and FIT generator for reuse
  - SoC family templates can use reused will adding Board specific variations
  - FIT generators can help runtime composing update of packing data.
- Familiarity when works across SoC vendors
  - More and more platforms are migrating to Binman
- Binman can inject symbols to final binary
  - Helps code at runtime detect and understand various image locations etc



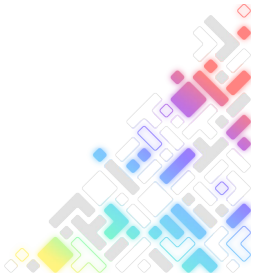
Future developments

- Ability to parse firmware names via CLI argument
  - Today file name hardcoded in FIT description
- Extending to other devices
  - Current boards using custom scripts that can benefit from Binman:
    - See the **tools/ folder** of U-Boot source for more such boards
- Signing within a FIT generator



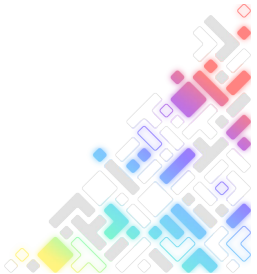
Open Discussion points

1. Move basic SPL and U-Boot image generation to Binman flow
    a. Stream lines packing and blob checks
2. Making this common tool across SoCs and across bootloader.
    a. Avoids duplications and eases device porting to multiple bootloaders
    b. Barebox, Coreboot etc ?
3. Are there any other fancy images that we need to support?



References

- OSFC 2019 Binman Talk - Simon Glass
- K3 Migration to using Binman patch series
- U-Boot Open Source Project
- U-Boot Documentation - Binman
- Open Source Summit Europe 2022 Bootloaders-101 - Bryan Brattlof

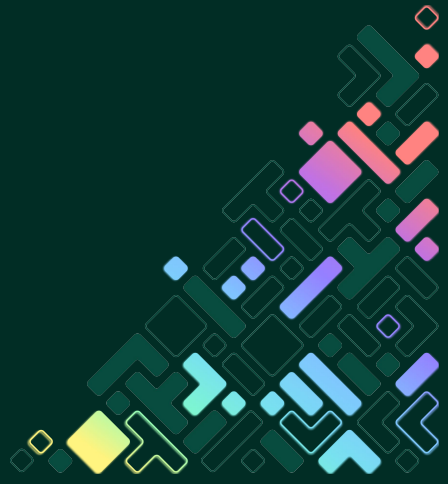


EMBEDDED
OPEN SOURCE
SUMMIT

Thank you!

Credits

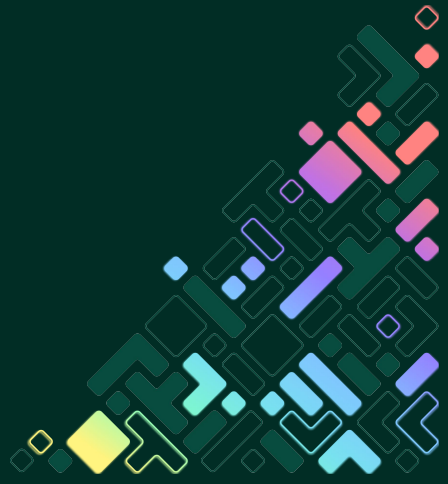
- Linux Foundation
- U-Boot community working on Binman ;)



EMBEDDED
OPEN SOURCE
SUMMIT

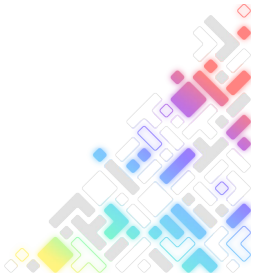
- Authors@:
  - Neha Malcom Francis n-francis@ti.com
  - Simon Glass sjg@chromium.com
  - Vignesh Raghavendra <vigneshr@ti.com>
- Also on IRC @ libera.chat #u-boot #linux-ti

Bonus Slides



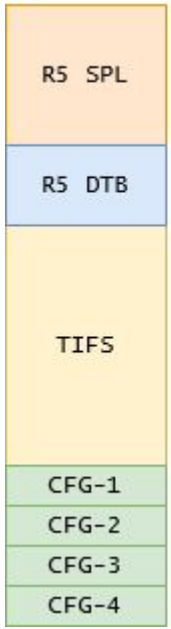
Bonus stuff

- R5 boot-loader image (Has much more complexity)
- New in Binman - Templating
- Binman - Runtime symbol updation



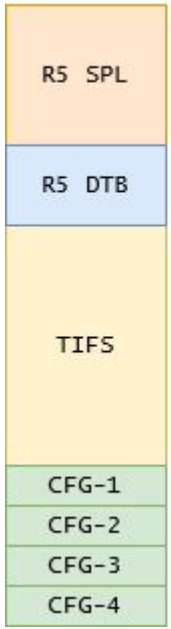
TI K3 Boot-loader binaries: a firmware tangle!

- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries



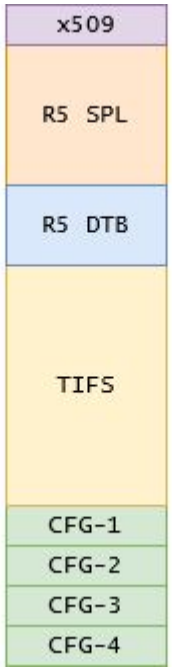
TI K3 Boot-loader binaries: a firmware tangle!

- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries
- Security?



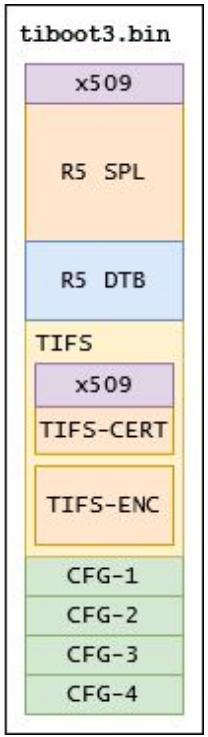
TI K3 Boot-loader binaries: a firmware tangle!

- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries
- Security?



TI K3 Boot-loader binaries: a firmware tangle!

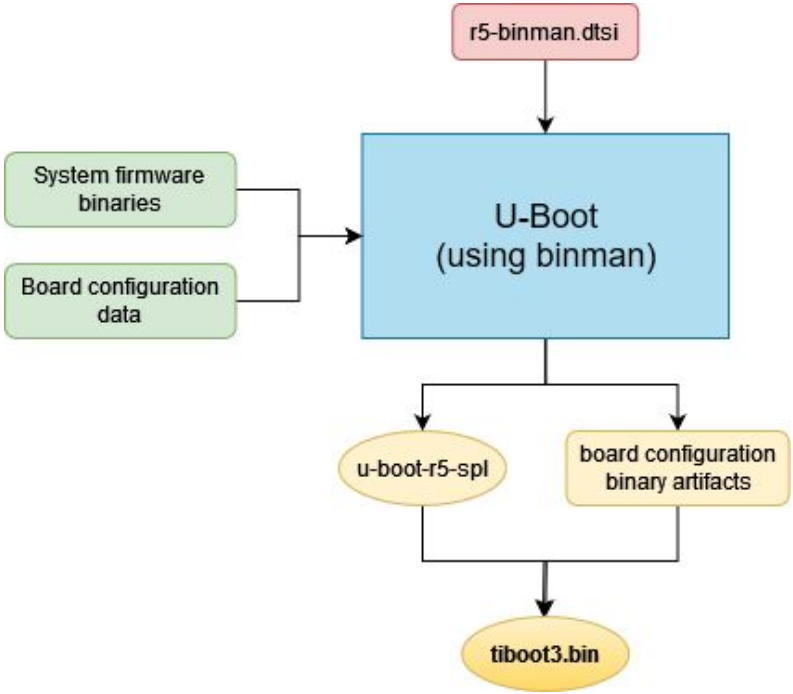
- TIFS (TI Foundational Security)
  - Platform security firmware
- Board configuration binaries
- Security? More?



TI K3 Boot-loader binaries: a firmware tangle!

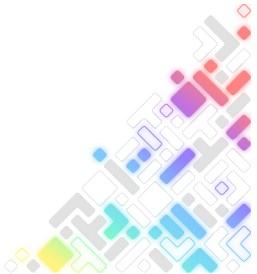
```
…
ifneq ($(SOC_TYPE),gp)
$(SYSFW_HS_CERTS_PATH): $(SYSFW_HS_INNER_CERT_PATH)
	@echo "Signing the SYSFW inner certificate with $(KEY) key... ";
	./gen_x509_cert.sh -d -c m3 -b $< -o $@ -l $(LOADADDR) -k $(KEY) -r $(SW_REV);

$(soc_objroot)/sysfw.bin-$(SOC_TYPE): $(SYSFW_HS_CERTS_PATH) $(SYSFW_PATH) | _objtree_build
	cat $^ > $@
else
$(soc_objroot)/sysfw.bin-$(SOC_TYPE): $(SYSFW_PATH) | _objtree_build
	@echo "Signing the SYSFW release image with $(KEY) key... ";
	./gen_x509_cert.sh -c m3 -b $< -o $@ -l $(LOADADDR) -k $(KEY) -r $(SW_REV);
endif

$(ITS): | _objtree_build
	./gen_its.sh $(SOC) $(SOC_TYPE) $(CONFIG) $(SOC_BINS) > $@

$(ITB): $(ITS) $(SOC_BINS) | _bindir_build
	$(MKIMAGE) -f $< -r $@

.PHONY: sysfw.itb
sysfw.itb: $(ITB)
	@ln -sf $< $(BIN_DIR)/$@

$(COMBINED_SYSFW_BRDCFG): $(soc_objroot)/board-cfg.bin $(soc_objroot)/sec-cfg.bin $(soc_objroot)/pm-cfg.bin $(soc_objroot)/rm-cfg.bin
	python3 ./scripts/sysfw_boardcfg_blob_creator.py -b $(soc_objroot)/board-cfg.bin -s $(soc_objroot)/sec-cfg.bin -p
$(soc_objroot)/pm-cfg.bin -r $(soc_objroot)/rm-cfg.bin -o $@

…
```


Extending Binman to support K3



New in Binman - Templates

- Common part in multiple images
- e.g. FIT with ATF, OPTEE etc.
- Reduce code redundancy



New in Binman - Templates

```
binman {
        u-boot.img {
                filename = “u-boot.img”;
                pad-byte = <0xff>;

                fit {
                        description = “U-Boot image”;

                        images {
                                uboot {
                                        type = “firmware”;
                                        os = “u-boot”;
                                        arch = “arm”;
                                        compression = “none”;
                                        load = <CONFIG_TEXT_BASE>;
                                        ti-secure {
                                                content = <&u_boot_nogtb>;
                                                Keyfile = “custMpk.pem”;
                                        };
                                        U_boot_nodtb: u-boot-nodtb {
                                        };
                                        Hash {
                                                Algoo = “crc32”;
                                        };
                                };
                                fdt-0 {
                                        description = “foo-board”;
                                        …
```

foo-board-binman.dtsi

New in Binman - Templates

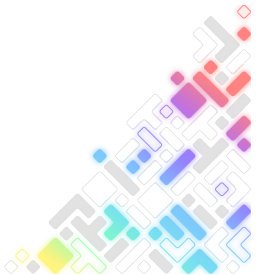
```
binman {
        uboot_template: u-boot.img {
                filename = “u-boot.img”;
                pad-byte = <0xff>;

                fit {
                        description = “U-Boot image”;

                        images {

                                uboot {

                                        type = “firmware”;
                                        os = “u-boot”;
                                        arch = “arm”;
                                        compression = “none”;
                                        load = <CONFIG_TEXT_BASE>;
                                        ti-secure {
                                                content = <&u_boot_nodtb>;
                                                Keyfile = “custMpk.pem”;
                                        };
                                        u_boot_nodtb: u-boot-nodtb {
                                        };
                                        Hash {
                                                Algoo = “crc32”;
                                        };
```


common.dtsi

New in Binman - Templates

```
#include “common.dtsi”
…
&binman {
        u-boot {
                insert-template = <&uboot_template>;
                fit {
                        images {
                                fdt-0 {
                                        description = “foo-board”;
                                        …
                                };
                        };
                };
        };
};
```

`foo-board-binman.dtsi`



Binman - Runtime symbol updation

```
binman_sym_declare(ulong, u_boot, image_pos);
…
void spl_set_header_raw_uboot(struct spl_image_info *spl_image)
{
        ulong u_boot_pos = binman_sym(ulong, u_boot_any, image_pos);
        spl_image->size = CONFIG_SYS_MONITOR_LEN;
        if (u_boot_pos && u_boot_pos != BINMAN_SYM_MISSING) {
                spl_image->entry_point = u_boot_pos;
                spl_image->load_addr = u_boot_pos;
        } else {
                spl_image->entry_point = CONFIG_SYS_UBOOT_START;
                spl_image->load_addr = CONFIG_SYS_TEXT_BASE;
        }
        spl_image->os = IH_OS_U_BOOT;
        spl_image->name = "U-Boot";
        …
```

Device Tree and U-Boot

- U-Boot supports Device Tree
  - Popular way of describing platform in embedded world
- Device tree
  - Data structure describing the hardware
  - Acyclic graph, made of named nodes containing properties

```
#include <dt-bindings/soc/ti,sci_pm_domain.h

/ {
      model = "Texas Instruments K3 J721E SOC";
      compatible = "ti,j721e";
      interrupt-parent = <&gic500>;
      #address-cells = <2>;
      #size-cells = <2>;
[...]
      cpus {
            #address-cells = <1>;
            #size-cells = <0>;
            cpu-map {
                  cluster0: cluster0 {
                        core0 {
                              cpu = <&cpu0>;
                        };
                        core1 {
                              cpu = <&cpu1>;
                        };
                  };
            };
            cpu0: cpu@0 {
                  compatible = "arm,cortex-a72";
                  reg = <0x000>;
                  device_type = "cpu";
[...]
```

https://www.devicetree.org/specifications/